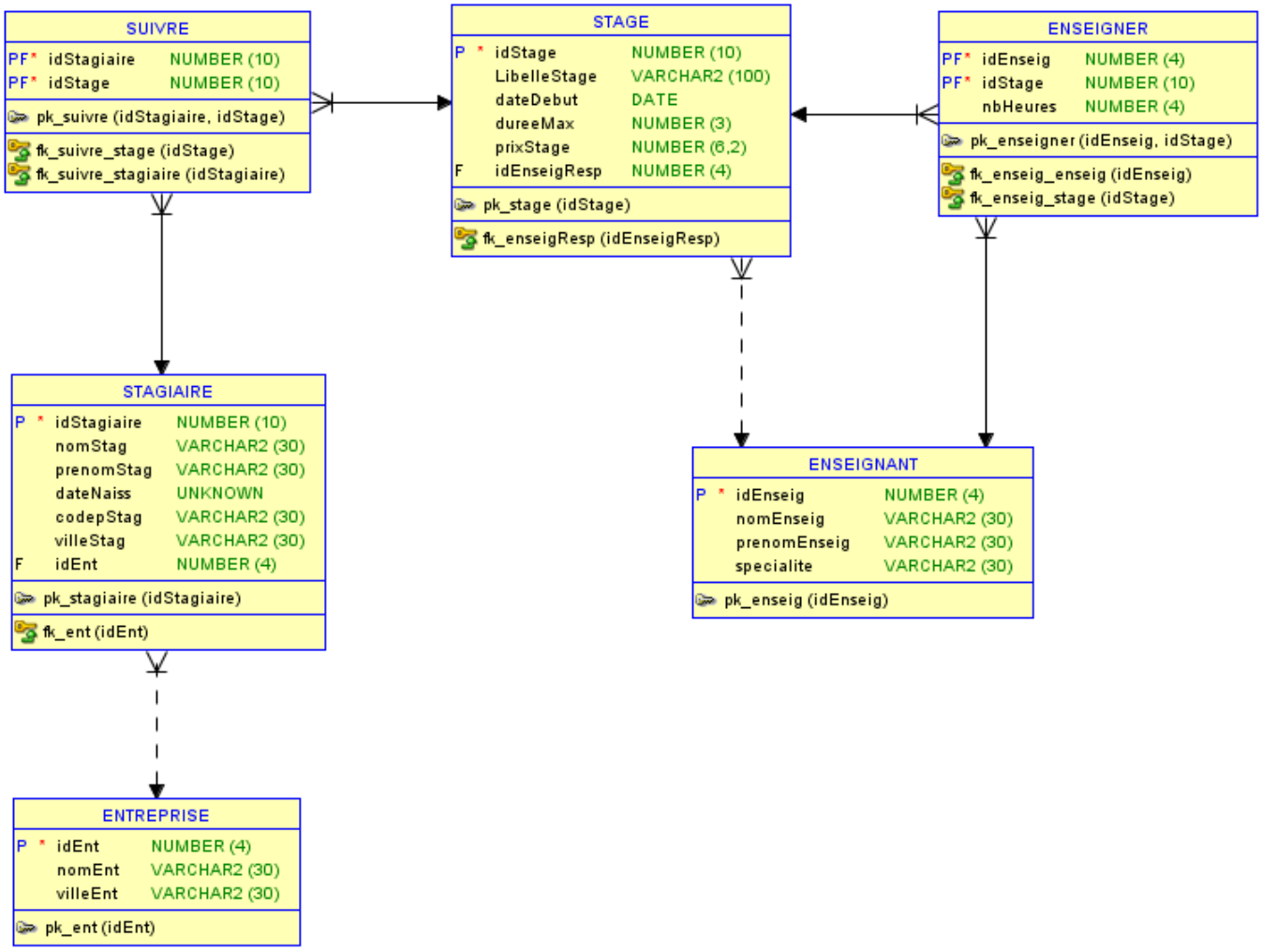
EISTI Ecole Internationale des Sciences du Traitement de l'Information	**EXAMEN DE** **TRAITEMENT ET ADMINISTRATION DES DONNEES** **Durée : 2 heures** **Tous les documents papiers sont autorisés**
Rédigé par : L'équipe pédagogique	Ref : *SE- TAD*
A l'intention de : Élèves ingénieurs 2ème année	Créé le : 08/12/2016

DCF est une société spécialisée dans l'organisation de stages professionnels pour les entreprises (formation continue). Elle accueille des stagiaires provenant de diverses entreprises qui en assurent le paiement. Plusieurs enseignants encadrent ces stages. Ils peuvent intervenir sur plusieurs stages. Les heures réalisées sont variables et dépendent des stages. Un enseignant responsable est désigné pour chaque stage. Ainsi, un enseignant peut-être responsable de plusieurs stages ou d'aucun.

Ces stages concernent l'informatique (logiciels), mais aussi la communication écrite ou orale, le management, … Un stagiaire peut suivre 1 ou plusieurs stages. On ne le rentre dans la base (et son entreprise) que lorsqu'il réalise effectivement un stage. La durée d'un stage est donnée en jours. Un jour correspond à 6 heures de stage.

La base de données de DCF est réalisée selon le schéma relationnel suivant. Vous trouverez en annexe un script SQL de création de la base. On suppose cette base créée sous Oracle dans le schéma « dcf » précédemment créé. « dcf » possède tous les droits d'administration sur l'ensemble de son schéma.
On suppose enfin que vous êtes connecté en tant que « system ».



I. Administration Base de Données (6 points)

1. Créer deux utilisateurs **scol** et **compta** avec le mot de passe par défaut DCF. **0.5pt**

```
CREATE USER scol IDENTIFIED BY scol DEFAULT TABLESPACE USERS;
CREATE USER compta IDENTIFIED BY compta DEFAULT TABLESPACE USERS;
```

2. Créer les rôles suivants :
    a) **RScol** : accès en lecture aux tables stage, enseigner, enseignant et tous les droits aux tables stagiaire, entreprise et suivre. **0.5pt**

```
CREATE ROLE Rscol;
GRANT connect TO RVendeur;
GRANT select ON dcf.stage TO RVendeur;
GRANT select ON dcf.enseigner TO RVendeur;
GRANT select ON dcf.enseignant TO RVendeur;
GRANT select, insert, update, delete ON dcf.stagiaire TO RVendeur;
GRANT select, insert, update, delete ON dcf.entreprise TO RVendeur;
GRANT select, insert, update, delete ON dcf.suivre TO RVendeur;
```

    b) **Rcompta** : accès en lecture à toutes les tables. **0.5pt**

```
CREATE ROLE Rcompta;
GRANT connect TO Rcompta;
GRANT select ON dcf.stage TO Rcompta;
GRANT select ON dcf.enseigner TO Rcompta;
GRANT select ON dcf.enseignant TO Rcompta;
GRANT select ON dcf.stagiaire TO Rcompta;
GRANT select ON dcf.entreprise TO Rcompta;
GRANT select ON dcf.suivre TO Rcompta;
```

3. Créer les vues suivantes : **1.5pt**
    a) La vue « **administration** » qui donne la liste des stages (tous les champs de la table avec les nom et prénom de l'enseignant responsable), le nombre d'élèves inscrits et le montant total généré. **1.5pt**

```
CREATE VIEW administration AS
      SELECT st.idStage, LibelleStage, dateDebut, dureeMax, idEnseigResp,
nomEnseig, prenomEnseig, COUNT(idStagiaire) NbStagiaires, SUM(prixStage)
CATotal – ou COUNT(idStagiaire)*prixStage
      FROM ENSEIGNANT e, STAGE st, SUIVRE su
      WHERE e.idEnseig = st.idEnseigResp
        AND st.idStage = su.idStage
      GROUP BY st.idStage, LibelleStage, dateDebut, dureeMax, idEnseigResp,
nomEnseig, prenomEnseig;
```

    b) La vue « **chargetravail** » qui permet de visualiser, par enseignant, la liste des stages pour lesquels il intervient (libellé, date de début et responsable), ainsi que le nombre d'heures correspondantes et le nombre d'élèves inscrits. **1.5pt**

```
CREATE OR REPLACE VIEW chargetravail AS
  SELECT et.idEnseig, st.LibelleStage, st.dateDebut, etresp.nomEnseig,
         er.nbHeures, count(sv.idStagiaire) NbStagiaires
  FROM enseignant et, enseignant etresp, enseigner er, stage st, suivre sv
  WHERE st.idEnseigResp = etresp.idEnseig
    AND st.idStage = er.idStage
    AND st.idStage = sv.idStage
    AND er.idEnseig = et.idEnseig
  GROUP BY et.idEnseig, st.LibelleStage, st.dateDebut, etresp.nomEnseig,
           er.nbHeures;
```

    c) La vue « **nowebdev** » qui donne la liste des entreprises (tous les champs) dans la base n'ayant pas de stagiaire qui ait suivi une formation « Développement Web ». **1.5pt**

```
CREATE OR REPLACE VIEW nowebdev AS
  SELECT idEnt, nomEnt, villeEnt
  FROM entreprise
  WHERE idEnt NOT IN (SELECT DISTINCT idEnt
                      FROM stagiaire sta, SUIVRE su, STAGE st
                      WHERE sta.idStagiaire = su.idStagiaire
                        AND su.idStage = st.idStage
                        AND LibelleStage = "Développement Web");
```

II. PL/SQL (6 points)

1. Pour quelles tables est-il utile de créer une séquence ? Écrire les séquences correspondantes. **1pt**

```
CREATE SEQUENCE entreprise START WITH 1
CREATE SEQUENCE stagiaire START WITH 1
CREATE SEQUENCE stage START WITH 1
CREATE SEQUENCE enseignant START WITH 1
```

2. Écrire une procédure qui édite une fiche récapitulative d'un stage contenant les informations suivantes : libellé, date de début, durée, tarif, nom et prénom de l'enseignant responsable et nombre de stagiaires inscrits. **2pts**

```
create or replace procedure ficheStage(ids  number)
is
  descStage  stage%rowtype;
  nbStagiaires number;
  descResponsable  enseignant%rowtype;
begin
  select * into descStage from stage where idstage = ids;
  dbms_output.put_line('Stage : '||descStage.LibelleStage);
  dbms_output.put_line('Date début : '||descStage.dateDebut);
  dbms_output.put_line('Durée : '||descStage.dureeMax||' jours');
  dbms_output.put_line('Tarif : '||descStage.prixStage);
  dbms_output.put_line('************************');

  select * into descResponsable from enseignant where
idEnseig=descStage.idEnseigResp;
  dbms_output.put_line('Responsable : '||descResponsable.prenomEnseig||'
'||descResponsable.nomEnseig);

  select count(*) into nbStagiaires from suivre where idStage=ids;
  dbms_output.put_line('Nombre de stagiaires : '||nbStagiaires);

EXCEPTION
when no_data_found then
 dbms_output.put_line('stage inexistant');
end ficheStage;
/
```

3. Écrire une fonction qui retourne la charge horaire d'un enseignant. On supposera que la responsabilité d'un stage occupe l'enseignant 2 heures par jour de formation. **1pt**

```
create or replace function volHoraireEns(idE  number) return number is
  nbhe number;
  nbjr number;
  res number;
begin
  select sum(nbHeures) into nbhe from enseigner where idEnseig=idE;
  select sum(dureeMax) into nbjr from stage where idEnseigResp=idE;
  res := nbhe+2*nbjr;
  return res;
end volHoraireEns;
/
```

4. Écrire une procédure qui affiche pour chaque enseignant son nom, son prénom et sa charge horaire. Cet affichage doit se terminer par celui de la charge horaire moyenne. **2pts**

```
create or replace procedure serviceEnseignants is
vh number;
nbe number;
sh number;
moyenne number(5,2);
cursor listeEnseignants is
  select * from enseignant;
begin
  nbe := 0;
  sh := 0;
  for ensg in listeEnseignants
  loop
    vh := volHoraireEns(ensg.idEnseig);
    dbms_output.put_line(ensg.nomEnseig||' '||ensg.prenomEnseig||' '||vh);
     nbe := nbe+1;
     sh := sh+vh;
  end loop;
  moyenne := sh/nbe;
  dbms_output.put_line('***************************');
  dbms_output.put_line('Moyenne: '||moyenne||' heures');
end serviceEnseignants;
/
```

III. JDBC (3 points)

Supposons que nous avons à notre disposition une classe Java MyConnectorJDBC pour établir une connexion à la base de données de DCF de la façon suivante :

```
import java.sql.*;
public class Dcf {


  public static void main(String [] args) {
        MyConnectorJDBC connector = new MyConnectorJDBC();

        try {
                MyConnectorJDBC.setParametersOracleLocal("system","password");
                Connection  connexion = connector.getConnection();
                System.out.println("Connexion OK");

                /* À compléter... */

                connexion.close();
                }
                catch(Exception e) {
                        e.printStackTrace();
            }
   } // fin main
} // fin classe Dcf
```

Réalisez les opérations suivantes :

1. Complétez le code de la classe Dcf pour obtenir le nom et le prénom des enseignants spécialistes d'une spécialité passée comme argument du programme exécuté en ligne de commande. **1pt**

```
String selectSQL = "SELECT idenseig,nomenseig,prenomenseig,specialite FROM
ENSEIGNANT WHERE LOWER(specialite) = LOWER(?)";
PreparedStatement req = connexion.prepareStatement(selectSQL);
req.setString(1,args[0]);
ResultSet rs = req.executeQuery();
System.out.println("Requete executée");

while (rs.next()) {
      int id = rs.getInt("idenseig");
      String nom = rs.getString("nomenseig");
      String prenom = rs.getString("prenomenseig");
      String specialite = rs.getString("specialite");
      System.out.println(id + "\t" + nom + "\t" + prenom + "\t" + specialite +
"\t");
}
```

2. Justifiez le type de requête utilisée dans la question 1 et expliquez la différence entre une requête SQL paramétrée et non paramétrée en Java. **1pt**

Une requête non paramétrée demande de créer directement une chaîne de caractère à partir des arguments et de la requête SQL. Elle n'est pas optimisée et doit être recompilée à chaque fois.
La requête paramétrée permet d'optimiser le type des arguments en utilisant leurs types de données de départ. De plus elle est compilée pour une réutilisation ultérieure.

3. Nous voulons ajouter au nom de chaque spécialité le suffix '_eisti'. Par exemple, si l'argument du programme est la spécialité 'Informatique', elle sera changée à 'Informatique_eisti' dans la base de données. Ajoutez le code pour mettre à jour les n-tuples concernés et montrez le nombre de n-tuples modifiés. **1pt**

```
String updateSQL = "UPDATE enseignant SET specialite = ?  WHERE
LOWER(specialite) = LOWER(?)";
req = connexion.prepareStatement(updateSQL);
req.setString(1,args[0]+"_eisti");
req.setString(2,args[0]);
int tuples = req.executeUpdate();
System.out.println("UPDATE executée avec "+tuples+" tuples modifiés");

// On pourrait aussi utiliser un ResultSet modifiable et maj specialité à
partir du ResultSet
Statement stmt = connexion.createStatement(
                        ResultSet.TYPE_SCROLL_INSENSITIVE,
                        ResultSet.CONCUR_UPDATABLE);
// ... et à l'intérieur du while (rs.next()) faire :
      specialite = rs.getString("specialite");
      rs.updateString("specialite", specialite+"_eisti");
      rs.updateRow();
```

IV. JAXP (5 points)

Soit le document entreprises.xml suivant :

```
<entreprises>
  <entreprise idEnt="1">
        <nomEnt>Turbomeca</nomEnt>
        <villeEnt>Bordeaux</villeEnt>
  </entreprise>
  <entreprise idEnt="2">
        <nomEnt>Thales</nomEnt>
        <villeEnt>Paris</villeEnt>
  </entreprise>
  <entreprise idEnt="3">
        <nomEnt>Zara</nomEnt>
        <villeEnt>Madrid</villeEnt>
  </entreprise>
  <entreprise idEnt="4">
        <nomEnt>Gucci</nomEnt>
        <villeEnt>Paris</villeEnt>
  </entreprise>
</entreprises>
```

1. Créez une classe Java gestionEntreprise et une fonction chargerXML qui charge un fichier XML comme un arbre DOM. Créez la fonction main() pour lire le fichier entreprises.xml. **1pt**

```
public class gestionEntreprise {

  public Document chargerXML(String filename) {
    Document doc;
    try {
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      //Pour obtenir un arbre DOM de niveau 2 avec des nodes qui
      //contiennent l'information du namespace.
      dbf.setNamespaceAware(true);

      DocumentBuilder db = dbf.newDocumentBuilder();
      doc = db.parse(new File(filename));
    }
    catch(Exception e) {
      return(null);
    }
    return doc;
  }
  public static void main(String[] args) throws Exception {
    String filename = "./entreprises.xml";
    Document entreprises;
    gestionEntreprise ge = new gestionEntreprise();
    entreprises = ge.chargerXML(filename);
  }
```

2. Ajoutez une fonction mutationSiege() qui, sur l'arbre DOM, change à Pau la ville de toutes les entreprises qui sont actuellement à Bordeaux. **2pts**

```
public void mutationSiege(Document ent) {

  String oldVille = "Bordeaux";
  String newVille = "Pau";
  Node entrepriseN,villeN;
  int i,j;

  Element racine = ent.getDocumentElement();
  NodeList nl = racine.getChildNodes();

  for(i = 0; i < nl.getLength() ; i++) {
    Node n = nl.item(i);
    if (n.getNodeType()==Node.ELEMENT_NODE ) {
      Element e = (Element)n;
      String idEnt = e.getAttribute("idEnt");
      System.out.println("idEnt: "+idEnt);

      NodeList entrepriseNl = e.getChildNodes();
      for(j = 0; j < entrepriseNl.getLength() ; j++) {
        entrepriseN = entrepriseNl.item(j);
        if (entrepriseN.getNodeName() == "villeEnt") {
          villeN = entrepriseN.getFirstChild();
          String villeEnt = villeN.getNodeValue();
          System.out.println("villeEnt: " + villeEnt);
          if(villeEnt.equals("Bordeaux")) {
            villeN.setNodeValue("Pau");
            System.out.println("Siège mutée à " + villeN.getNodeValue());
          }
          System.out.println("nodevalue: " + villeN.getNodeValue());
        }
      }
    }
  }
}

public static void main(String[] args) throws Exception {
  String filename = "./entreprises.xml";
  String xsdfilename = "./entreprises.xsd";
  Document entreprises;

  ge.mutationSiege(entreprises);

}
```

3. Nous avons le fichier entreprises.xsd suivant pour valider notre document xml :

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="entreprises ">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="entreprise" minOccurs="10" maxOccurs="unbounded">
         <xs:complexType>
                        <xs:sequence>
                                <xs:element name="nomEnt"/>
                                <xs:element name="villeEnt"/>
                        </xs:sequence>
            <xs:attribute name="idEnt" type="xs:int"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Créez une fonction chargerXMLavecXsd() et une classe MyErrorHandler pour lire et valider le document entreprises.xml avec ce schéma. Quelle sera le résultat de la validation pour ce document XML ? Justifiez. **2pts**

```
  public Document chargerXMLavecXsd(String filename,String filenamexsd) {
    Document doc;
    String JAXP_SCHEMA_LANGUAGE =
"http://java.sun.com/xml/jaxp/properties/schemaLanguage";
    String W3C_XML_SCHEMA = "http://www.w3.org/2001/XMLSchema";
    String JAXP_SCHEMA_SOURCE
="http://java.sun.com/xml/jaxp/properties/schemaSource";

    try {
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

      MyErrorHandler errorHandler = new MyErrorHandler();
      dbf.setNamespaceAware(true);
      dbf.setValidating(true);
      dbf.setAttribute(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
      dbf.setAttribute(JAXP_SCHEMA_SOURCE, new File(filenamexsd));
      DocumentBuilder db = dbf.newDocumentBuilder();
      db.setErrorHandler(errorHandler);
      doc = db.parse(new File(filename));
    } catch (ParserConfigurationException e) {
      System.err.println(e.getMessage());
      return null;
    } catch (SAXException e) {
      System.err.println(e.getMessage());
      return null;
    } catch (java.io.IOException e) {
      System.err.println(e.getMessage());
      return null;
    }
    System.out.println("Validation : Document valide");
    return doc;
  }

class MyErrorHandler implements ErrorHandler {

  public void warning(SAXParseException spe) throws SAXException {
    System.err.println("Warning: " + spe.getMessage());
  }

  public void error(SAXParseException spe) throws SAXException {
    String message = "Error: " + spe.getMessage();
    throw new SAXException(message);
  }

  public void fatalError(SAXParseException spe) throws SAXException {
    String message = "Fatal Error: " + spe.getMessage();
    throw new SAXException(message);
  }
}
```

ANNEXES : Script SQL de création de la base de données DCF :

```
CREATE TABLE ENSEIGNANT (
    idEnseig NUMBER(4) NOT NULL CONSTRAINT pk_enseig PRIMARY KEY,
    nomEnseig VARCHAR2(30),
    prenomEnseig VARCHAR2(30),
    specialite VARCHAR2(30)
);

CREATE TABLE STAGE (
    idStage NUMBER(10) NOT NULL CONSTRAINT pk_stage PRIMARY KEY,
    LibelleStage VARCHAR2(100),
    dateDebut DATE,
    dureeMax NUMBER(3),  -- nombre de jours, sachant qu'un jour contient 6 heures de formation
    prixStage NUMBER(6,2),
    idEnseigResp NUMBER(4),
    CONSTRAINT fk_enseigResp FOREIGN KEY(idEnseigResp) REFERENCES ENSEIGNANT(idEnseig)
);

CREATE TABLE ENTREPRISE (
    idEnt NUMBER(4) NOT NULL CONSTRAINT pk_ent PRIMARY KEY,
    nomEnt VARCHAR2(30),
    villeEnt VARCHAR2(30)
);

CREATE TABLE STAGIAIRE (
    idStagiaire NUMBER(10) NOT NULL CONSTRAINT pk_stagiaire PRIMARY KEY,
    nomStag VARCHAR2(30),
    prenomStag VARCHAR2(30),
    dateNaiss DATE,
    rueStag VARCHAR2(30),
    codepStag VARCHAR2(30),
    villeStag VARCHAR2(30),
    idEnt NUMBER(4),
    CONSTRAINT fk_ent FOREIGN KEY(idEnt) REFERENCES ENTREPRISE(idEnt)
);

CREATE TABLE ENSEIGNER (
    idEnseig NUMBER(4) NOT NULL,
    idStage NUMBER(10) NOT NULL,
    nbHeures NUMBER(4),
    CONSTRAINT pk_enseigner PRIMARY KEY (idEnseig, idStage),
    CONSTRAINT fk_enseig_enseig FOREIGN KEY(idEnseig) REFERENCES ENSEIGNANT(idEnseig),
    CONSTRAINT fk_enseig_stage FOREIGN KEY(idStage) REFERENCES STAGE(idStage)
);

CREATE TABLE SUIVRE (
    idStagiaire NUMBER(10) NOT NULL,
    idStage NUMBER(10) NOT NULL,
    CONSTRAINT pk_suivre PRIMARY KEY (idStagiaire, idStage),
    CONSTRAINT fk_suivre_stagiaire FOREIGN KEY(idStagiaire) REFERENCES STAGIAIRE(idStagiaire),
    CONSTRAINT fk_suivre_stage FOREIGN KEY(idStage) REFERENCES STAGE(idStage)
);
```